Which GPU(s) to Get for Deep Learning: My Experience and Advice for Using GPUs in Deep Learning

2023-01-30 by [Tim Dettmers](https://timdettmers.com/author/tim-dettmers/) [1,663 Comments](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#comments)

Deep learning is a field with intense computational requirements, and your choice of GPU will fundamentally determine your deep learning experience. But what features are important if you want to buy a new GPU? GPU RAM, cores, tensor cores, caches? How to make a cost-efficient choice? This blog post will delve into these questions, tackle common misconceptions, give you an intuitive understanding of how to think about GPUs, and will lend you advice, which will help you to make a choice that is right for you.

This blog post is designed to give you different levels of understanding of GPUs and the new Ampere series GPUs from NVIDIA. You have the choice: (1) If you are not interested in the details of how GPUs work, what makes a GPU fast compared to a CPU, and what is unique about the new NVIDIA RTX 40 Ampere series, you can skip right to the performance and performance per dollar charts and the recommendation section. The cost/performance numbers form the core of the blog post and the content surrounding it explains the details of what makes up GPU performance.

(2) If you worry about specific questions, I have answered and addressed the most common questions and misconceptions in the later part of the blog post.

(3) If you want to get an in-depth understanding of how GPUs, caches, and Tensor Cores work, the best is to read the blog post from start to finish. You might want to skip a section or two based on your understanding of the presented topics.

**Contents**  [hide](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/)

[Overview](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Overview)

[How do GPUs work?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#How_do_GPUs_work)

[The Most Important GPU Specs for Deep Learning Processing Speed](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#The_Most_Important_GPU_Specs_for_Deep_Learning_Processing_Speed)

[Tensor Cores](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Tensor_Cores)

[Matrix multiplication without Tensor Cores](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Matrix_multiplication_without_Tensor_Cores)

[Matrix multiplication with Tensor Cores](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Matrix_multiplication_with_Tensor_Cores)

[Matrix multiplication with Tensor Cores and Asynchronous copies (RTX 30/RTX 40) and TMA (H100)](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Matrix_multiplication_with_Tensor_Cores_and_Asynchronous_copies_RTX_30RTX_40_and_TMA_H100)

[Memory Bandwidth](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Memory_Bandwidth)

[L2 Cache / Shared Memory / L1 Cache / Registers](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#L2_Cache_Shared_Memory_L1_Cache_Registers)

[Estimating Ada / Hopper Deep Learning Performance](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Estimating_Ada_Hopper_Deep_Learning_Performance)

[Practical Ada / Hopper Speed Estimates](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Practical_Ada_Hopper_Speed_Estimates)

[Possible Biases in Estimates](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Possible_Biases_in_Estimates)

[Advantages and Problems for RTX40 and RTX 30 Series](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Advantages_and_Problems_for_RTX40_and_RTX_30_Series)

[Sparse Network Training](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Sparse_Network_Training)

[Low-precision Computation](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Low-precision_Computation)

[Fan Designs and GPUs Temperature Issues](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Fan_Designs_and_GPUs_Temperature_Issues)

[3-slot Design and Power Issues](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#3-slot_Design_and_Power_Issues)

[Power Limiting: An Elegant Solution to Solve the Power Problem?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Power_Limiting_An_Elegant_Solution_to_Solve_the_Power_Problem)

[RTX 4090s and Melting Power Connectors: How to Prevent Problems](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#RTX_4090s_and_Melting_Power_Connectors_How_to_Prevent_Problems)

[8-bit Float Support in H100 and RTX 40 series GPUs](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#8-bit_Float_Support_in_H100_and_RTX_40_series_GPUs)

[Raw Performance Ranking of GPUs](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Raw_Performance_Ranking_of_GPUs)

[GPU Deep Learning Performance per Dollar](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#GPU_Deep_Learning_Performance_per_Dollar)

[GPU Recommendations](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#GPU_Recommendations)

[Is it better to wait for future GPUs for an upgrade? The future of GPUs.](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Is_it_better_to_wait_for_future_GPUs_for_an_upgrade_The_future_of_GPUs)

[Question & Answers & Misconceptions](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Question_Answers_Misconceptions)

[Do I need PCIe 4.0 or PCIe 5.0?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Do_I_need_PCIe_40_or_PCIe_50)

[Do I need 8x/16x PCIe lanes?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Do_I_need_8x16x_PCIe_lanes)

[How do I fit 4x RTX 4090 or 3090 if they take up 3 PCIe slots each?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#How_do_I_fit_4x_RTX_4090_or_3090_if_they_take_up_3_PCIe_slots_each)

[How do I cool 4x RTX 3090 or 4x RTX 3080?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#How_do_I_cool_4x_RTX_3090_or_4x_RTX_3080)

[Can I use multiple GPUs of different GPU types?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Can_I_use_multiple_GPUs_of_different_GPU_types)

[What is NVLink, and is it useful?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#What_is_NVLink_and_is_it_useful)

[I do not have enough money, even for the cheapest GPUs you recommend. What can I do?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#I_do_not_have_enough_money_even_for_the_cheapest_GPUs_you_recommend_What_can_I_do)

[What is the carbon footprint of GPUs? How can I use GPUs without polluting the environment?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#What_is_the_carbon_footprint_of_GPUs_How_can_I_use_GPUs_without_polluting_the_environment)

[What do I need to parallelize across two machines?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#What_do_I_need_to_parallelize_across_two_machines)

[Is the sparse matrix multiplication features suitable for sparse matrices in general?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Is_the_sparse_matrix_multiplication_features_suitable_for_sparse_matrices_in_general)

[Do I need an Intel CPU to power a multi-GPU setup?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Do_I_need_an_Intel_CPU_to_power_a_multi-GPU_setup)

[Does computer case design matter for cooling?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Does_computer_case_design_matter_for_cooling)

[Will AMD GPUs + ROCm ever catch up with NVIDIA GPUs + CUDA?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Will_AMD_GPUs_ROCm_ever_catch_up_with_NVIDIA_GPUs_CUDA)

[When is it better to use the cloud vs a dedicated GPU desktop/server?](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#When_is_it_better_to_use_the_cloud_vs_a_dedicated_GPU_desktopserver)

[Version History](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Version_History)

[Acknowledgments](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Acknowledgments)

[Related](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Related)

[Related Posts](https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/#Related_Posts)

**Overview**

This blog post is structured in the following way. First, I will explain what makes a GPU fast. I will discuss CPUs vs GPUs, Tensor Cores, memory bandwidth, and the memory hierarchy of GPUs and how these relate to deep learning performance. These explanations might help you get a more intuitive sense of what to look for in a GPU. I discuss the unique features of the new NVIDIA RTX 40 Ampere GPU series that are worth considering if you buy a GPU. From there, I make GPU recommendations for different scenarios. After that follows a Q&A section of common questions posed to me in Twitter threads; in that section, I will also address common misconceptions and some miscellaneous issues, such as cloud vs desktop, cooling, AMD vs NVIDIA, and others.

How do GPUs work?

If you use GPUs frequently, it is useful to understand how they work. This knowledge will help you to undstand cases where are GPUs fast or slow. In turn, you might be able to understand better why you need a GPU in the first place and how other future hardware options might be able to compete. You can skip this section if you just want the useful performance numbers and arguments to help you decide which GPU to buy. The best high-level explanation for the question of how GPUs work is my following Quora answer:

Read [Tim Dettmers](https://www.quora.com/Tim-Dettmers-1)‘ [~~answer~~](https://timdettmers.com/Why-are-GPUs-well-suited-to-deep-learning?top_ans=21379913) to [~~Why are GPUs well-suited to deep learning?~~](https://timdettmers.com/Why-are-GPUs-well-suited-to-deep-learning) on [Quora](https://www.quora.com/)

This is a high-level explanation that explains quite well why GPUs are better than CPUs for deep learning. If we look at the details, we can understand what makes one GPU better than another.

The Most Important GPU Specs for Deep Learning Processing Speed

This section can help you build a more intuitive understanding of how to think about deep learning performance. This understanding will help you to evaluate future GPUs by yourself. This section is sorted by the importance of each component. Tensor Cores are most important, followed by memory bandwidth of a GPU, the cache hierachy, and only then FLOPS of a GPU.

Tensor Cores

Tensor Cores are tiny cores that perform very efficient matrix multiplication. Since the most expensive part of any deep neural network is matrix multiplication Tensor Cores are very useful. In fast, they are so powerful, that I do not recommend any GPUs that do not have Tensor Cores.

It is helpful to understand how they work to appreciate the importance of these computational units specialized for matrix multiplication. Here I will show you a simple example of A\*B=C matrix multiplication, where all matrices have a size of 32×32, what a computational pattern looks like with and without Tensor Cores. This is a simplified example, and not the exact way how a high performing matrix multiplication kernel would be written, but it has all the basics. A CUDA programmer would take this as a first “draft” and then optimize it step-by-step with concepts like double buffering, register optimization, occupancy optimization, instruction-level parallelism, and many others, which I will not discuss at this point.

To understand this example fully, you have to understand the concepts of cycles. If a processor runs at 1GHz, it can do 10^9 cycles per second. Each cycle represents an opportunity for computation. However, most of the time, operations take longer than one cycle. Thus we essentially have a queue where the next operations needs to wait for the next operation to finish. This is also called the latency of the operation.

Here are some important latency cycle timings for operations. These times can change from GPU generation to GPU generation. [These numbers are for Ampere GPUs](https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s33322/), which have relatively slow caches.

* Global memory access (up to 80GB): ~380 cycles
* L2 cache: ~200 cycles
* L1 cache or Shared memory access (up to 128 kb per Streaming Multiprocessor): ~34 cycles
* Fused multiplication and addition, a\*b+c (FFMA): 4 cycles
* Tensor Core matrix multiply: 1 cycle

Each operation is always performed by a pack of 32 threads. This pack is termed a warp of threads. Warps usually operate in a synchronous pattern — threads within a warp have to wait for each other. All memory operations on the GPU are optimized for warps. For example, loading from global memory happens at a granularity of 32\*4 bytes, exactly 32 floats, exactly one float for each thread in a warp. We can have up to 32 warps = 1024 threads in a streaming multiprocessor (SM), the GPU-equivalent of a CPU core. The resources of an SM are divided up among all active warps. This means that sometimes we want to run fewer warps to have more registers/shared memory/Tensor Core resources per warp.

For both of the following examples, we assume we have the same computational resources. For this small example of a 32×32 matrix multiply, we use 8 SMs (about 10% of an RTX 3090) and 8 warps per SM.

To understand how the cycle latencies play together with resources like threads per SM and shared memory per SM, we now look at examples of matrix multiplication. While the following example roughly follows the sequence of computational steps of matrix multiplication for both with and without Tensor Cores, please note that these are very simplified examples. Real cases of matrix multiplication involve much larger shared memory tiles and slightly different computational patterns.

Matrix multiplication without Tensor Cores

If we want to do an A\*B=C matrix multiply, where each matrix is of size 32×32, then we want to load memory that we repeatedly access into shared memory because its latency is about five times lower (200 cycles vs 34 cycles). A memory block in shared memory is often referred to as a memory tile or just a tile. Loading two 32×32 floats into a shared memory tile can happen in parallel by using 2\*32 warps. We have 8 SMs with 8 warps each, so due to parallelization, we only need to do a single sequential load from global to shared memory, which takes 200 cycles.

To do the matrix multiplication, we now need to load a vector of 32 numbers from shared memory A and shared memory B and perform a fused multiply-and-accumulate (FFMA). Then store the outputs in registers C. We divide the work so that each SM does 8x dot products (32×32) to compute 8 outputs of C. Why this is exactly 8 (4 in older algorithms) is very technical. I recommend Scott Gray’s blog post on [matrix multiplication](https://github.com/NervanaSystems/maxas/wiki/SGEMM) to understand this. This means we have 8x shared memory accesses at the cost of 34 cycles each and 8 FFMA operations (32 in parallel), which cost 4 cycles each. In total, we thus have a cost of:

200 cycles (global memory) + 8\*34 cycles (shared memory) + 8\*4 cycles (FFMA) = 504 cycles

Let’s look at the cycle cost of using Tensor Cores.

Matrix multiplication with Tensor Cores

With Tensor Cores, we can perform a 4×4 matrix multiplication in one cycle. To do that, we first need to get memory into the Tensor Core. Similarly to the above, we need to read from global memory (200 cycles) and store in shared memory. To do a 32×32 matrix multiply, we need to do 8×8=64 Tensor Cores operations. A single SM has 8 Tensor Cores. So with 8 SMs, we have 64 Tensor Cores — just the number that we need! We can transfer the data from shared memory to the Tensor Cores with 1 memory transfers (34 cycles) and then do those 64 parallel Tensor Core operations (1 cycle). This means the total cost for Tensor Cores matrix multiplication, in this case, is:

200 cycles (global memory) + 34 cycles (shared memory) + 1 cycle (Tensor Core) = 235 cycles.

Thus we reduce the matrix multiplication cost significantly from 504 cycles to 235 cycles via Tensor Cores. In this simplified case, the Tensor Cores reduced the cost of both shared memory access and FFMA operations.

This example is simplified, for example, usually each thread needs to calculate which memory to read and write to as you transfer data from global memory to shared memory. With the new Hooper (H100) architectures we additionally have the Tensor Memory Accelerator (TMA) compute these indices in hardware and thus help each thread to focus on more computation rather than computing indices.

Matrix multiplication with Tensor Cores and Asynchronous copies (RTX 30/RTX 40) and TMA (H100)

The RTX 30 Ampere and RTX 40 Ada series GPUs additionally have support to perform asynchronous transfers between global and shared memory. The H100 Hopper GPU extends this further by introducing the Tensor Memory Accelerator (TMA) unit. the TMA unit combines asynchronous copies and index calculation for read and writes simultaneously — so each thread no longer needs to calculate which is the next element to read and each thread can focus on doing more matrix multiplication calculations. This looks as follows.

The TMA unit fetches memory from global to shared memory (200 cycles). Once the data arrives, the TMA unit fetches the next block of data asynchronously from global memory. While this is happening, the threads load data from shared memory and perform the matrix multiplication via the tensor core. Once the threads are finished they wait for the TMA unit to finish the next data transfer, and the sequence repeats.

As such, due to the asynchronous nature, the second global memory read by the TMA unit is already progressing as the threads process the current shared memory tile. This means, the second read takes only 200 – 34 – 1 = 165 cycles.

Since we do many reads, only the first memory access will be slow and all other memory accesses will be partially overlapped with the TMA unit. Thus on average, we reduce the time by 35 cycles.

165 cycles (wait for async copy to finish) + 34 cycles (shared memory) + 1 cycle (Tensor Core) = 200 cycles.

Which accelerates the matrix multiplication by another 15%.

From these examples, it becomes clear why the next attribute, memory bandwidth, is so crucial for Tensor-Core-equipped GPUs. Since global memory is the by far the largest cycle cost for matrix multiplication with Tensor Cores, we would even have faster GPUs if the global memory latency could be reduced. We can do this by either increasing the clock frequency of the memory (more cycles per second, but also more heat and higher energy requirements) or by increasing the number of elements that can be transferred at any one time (bus width).

Memory Bandwidth

From the previous section, we have seen that Tensor Cores are very fast. So fast, in fact, that they are idle most of the time as they are waiting for memory to arrive from global memory. For example, during GPT-3-sized training, which uses huge matrices — the larger, the better for Tensor Cores — we have a Tensor Core TFLOPS utilization of about 45-65%, meaning that even for the large neural networks about 50% of the time, Tensor Cores are idle.

This means that when comparing two GPUs with Tensor Cores, one of the single best indicators for each GPU’s performance is their memory bandwidth. For example, The A100 GPU has 1,555 GB/s memory bandwidth vs the 900 GB/s of the V100. As such, a basic estimate of speedup of an A100 vs V100 is 1555/900 = 1.73x.

L2 Cache / Shared Memory / L1 Cache / Registers

Since memory transfers to the Tensor Cores are the limiting factor in performance, we are looking for other GPU attributes that enable faster memory transfer to Tensor Cores. L2 cache, shared memory, L1 cache, and amount of registers used are all related. To understand how a memory hierarchy enables faster memory transfers, it helps to understand how matrix multiplication is performed on a GPU.

To perform matrix multiplication, we exploit the memory hierarchy of a GPU that goes from slow global memory, to faster L2 memory, to fast local shared memory, to lightning-fast registers. However, the faster the memory, the smaller it is.

While logically, L2 and L1 memory are the same, L2 cache is larger and thus the average physical distance that need to be traversed to retrieve a cache line is larger. You can see the L1 and L2 caches as organized warehouses where you want to retrieve an item. You know where the item is, but to go there takes on average much longer for the larger warehouse. This is the essential difference between L1 and L2 caches. Large = slow, small = fast.

For matrix multiplication we can use this hierarchical separate into smaller and smaller and thus faster and faster chunks of memory to perform very fast matrix multiplications. For that, we need to chunk the big matrix multiplication into smaller sub-matrix multiplications. These chunks are called memory tiles, or often for short just tiles.

We perform matrix multiplication across these smaller tiles in local shared memory that is fast and close to the streaming multiprocessor (SM) — the equivalent of a CPU core. With Tensor Cores, we go a step further: We take each tile and load a part of these tiles into Tensor Cores which is directly addressed by registers. A matrix memory tile in L2 cache is 3-5x faster than global GPU memory (GPU RAM), shared memory is ~7-10x faster than the global GPU memory, whereas the Tensor Cores’ registers are ~200x faster than the global GPU memory.

Having larger tiles means we can reuse more memory. I wrote about this in detail in my [TPU vs GPU](https://timdettmers.com/2018/10/17/tpus-vs-gpus-for-transformers-bert/) blog post. In fact, you can see TPUs as having very, very, large tiles for each Tensor Core. As such, TPUs can reuse much more memory with each transfer from global memory, which makes them a little bit more efficient at matrix multiplications than GPUs.

Each tile size is determined by how much memory we have per streaming multiprocessor (SM) and how much we L2 cache we have across all SMs. We have the following shared memory sizes on the following architectures:

* Volta (Titan V): 128kb shared memory / 6 MB L2
* Turing (RTX 20s series): 96 kb shared memory / 5.5 MB L2
* Ampere (RTX 30s series): 128 kb shared memory / 6 MB L2
* Ada (RTX 40s series): 128 kb shared memory / 72 MB L2

We see that Ada has a much larger L2 cache allowing for larger tile sizes, which reduces global memory access. For example, for BERT large during training, the input and weight matrix of any matrix multiplication fit neatly into the L2 cache of Ada (but not other Us). As such, data needs to be loaded from global memory only once and then data is available throught the L2 cache, making matrix multiplication about 1.5 – 2.0x faster for this architecture for Ada. For larger models the speedups are lower during training but certain sweetspots exist which may make certain models much faster. Inference, with a batch size larger than 8 can also benefit immensely from the larger L2 caches.

Estimating Ada / Hopper Deep Learning Performance

This section is for those who want to understand the more technical details of how I derive the performance estimates for Ampere GPUs. If you do not care about these technical aspects, it is safe to skip this section.

Practical Ada / Hopper Speed Estimates

Suppose we have an estimate for one GPU of a GPU-architecture like Hopper, Ada, Ampere, Turing, or Volta. It is easy to extrapolate these results to other GPUs from the same architecture/series. Luckily, NVIDIA already [benchmarked the A100 vs V100 vs H100](https://developer.nvidia.com/deep-learning-performance-training-inference) across a wide range of computer vision and natural language understanding tasks. Unfortunately, NVIDIA made sure that these numbers are not directly comparable by using different batch sizes and the number of GPUs whenever possible to favor results for the H100 GPU. So in a sense, the benchmark numbers are partially honest, partially marketing numbers. In general, you could argue that using larger batch sizes is fair, as the H100/A100 GPU has more memory. Still, to compare GPU architectures, we should evaluate unbiased memory performance with the same batch size.

To get an unbiased estimate, we can scale the data center GPU results in two ways: (1) account for the differences in batch size, (2) account for the differences in using 1 vs 8 GPUs. We are lucky that we can find such an estimate for both biases in the data that NVIDIA provides.

Doubling the batch size increases throughput in terms of images/s (CNNs) by 13.6%. I benchmarked the same problem for transformers on my RTX Titan and found, surprisingly, the very same result: 13.5% — it appears that this is a robust estimate.

As we parallelize networks across more and more GPUs, we lose performance due to some networking overhead. The A100 8x GPU system has better networking (NVLink 3.0) than the V100 8x GPU system (NVLink 2.0) — this is another confounding factor. Looking directly at the data from NVIDIA, we can find that for CNNs, a system with 8x A100 has a 5% lower overhead than a system of 8x V100. This means if going from 1x A100 to 8x A100 gives you a speedup of, say, 7.00x, then going from 1x V100 to 8x V100 only gives you a speedup of 6.67x.  For transformers, the figure is 7%.

Using these figures, we can estimate the speedup for a few specific deep learning architectures from the direct data that NVIDIA provides. The Tesla A100 offers the following speedup over the Tesla V100:

* SE-ResNeXt101: 1.43x
* Masked-R-CNN: 1.47x
* Transformer (12 layer, Machine Translation, WMT14 en-de): 1.70x

Thus, the figures are a bit lower than the theoretical estimate for computer vision. This might be due to smaller tensor dimensions, overhead from operations that are needed to prepare the matrix multiplication like img2col or Fast Fourier Transform (FFT), or operations that cannot saturate the GPU (final layers are often relatively small). It could also be artifacts of the specific architectures (grouped convolution).

The practical transformer estimate is very close to the theoretical estimate. This is probably because algorithms for huge matrices are very straightforward. I will use these practical estimates to calculate the cost efficiency of GPUs.

Possible Biases in Estimates

The estimates above are for H100, A100 , and V100 GPUs. In the past, NVIDIA sneaked unannounced performance degradations into the “gaming” RTX GPUs: (1) Decreased Tensor Core utilization, (2) gaming fans for cooling, (3) disabled peer-to-peer GPU transfers. It might be possible that there are unannounced performance degradations in the RTX 40 series compared to the full Hopper H100.

As of now, one of these degradations was found for Ampere GPUs: Tensor Core performance was decreased so that RTX 30 series GPUs are not as good as Quadro cards for deep learning purposes. This was also done for the RTX 20 series, so it is nothing new, but this time it was also done for the Titan equivalent card, the RTX 3090. The RTX Titan did not have performance degradation enabled.

Currently, no degradation for Ada GPUs are known, but I update this post with news on this and let my followers on [twitter](https://twitter.com/Tim_Dettmers) know.

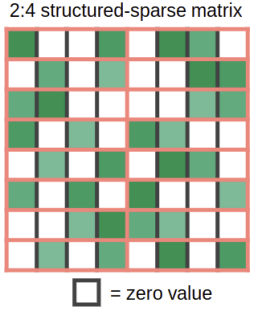
**Advantages and Problems for RTX40 and RTX 30 Series**

The new NVIDIA Ampere RTX 30 series has additional benefits over the NVIDIA Turing RTX 20 series, such as sparse network training and inference. Other features, such as the new data types, should be seen more as an ease-of-use-feature as they provide the same performance boost as Turing does but without any extra programming required.

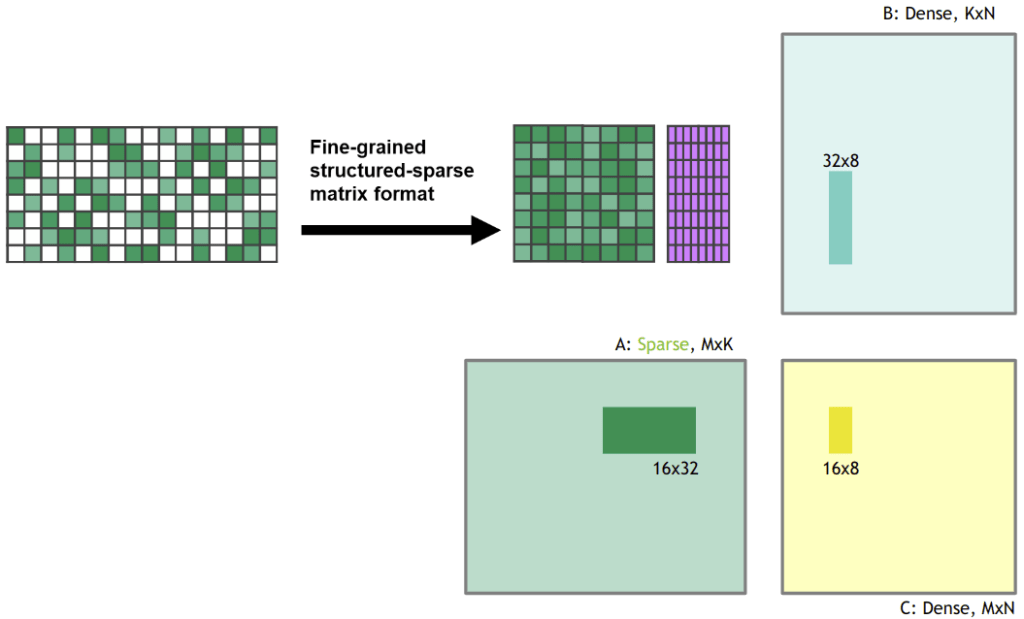
The Ada RTX 40 series has even further advances like 8-bit Float (FP8) tensor cores. The RTX 40 series also has similar power and temperature issues compared to the RTX 30. The issue of melting power connector cables in the RTX 40 can be easily prevented by connecting the power cable correctly.

Sparse Network Training

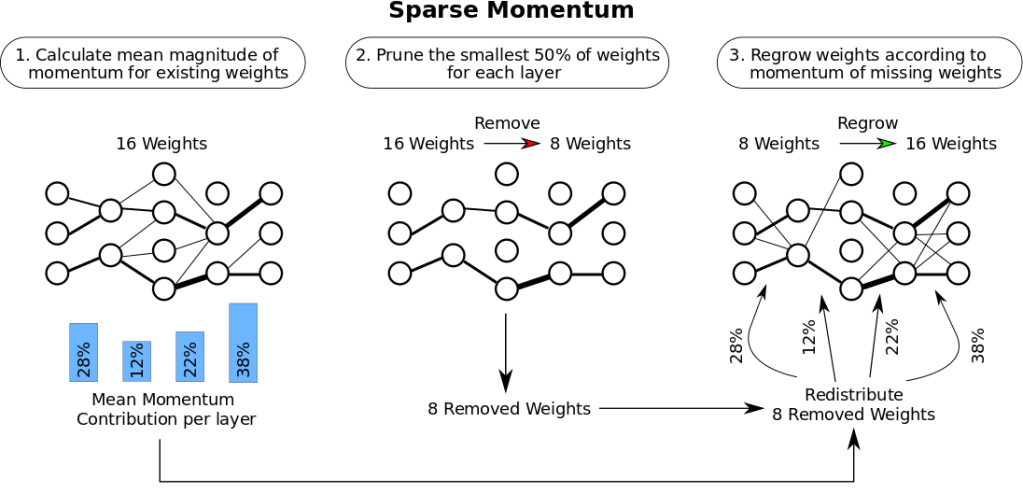
Ampere allows for fine-grained structure automatic sparse matrix multiplication at dense speeds. How does this work? Take a weight matrix and slice it into pieces of 4 elements. Now imagine 2 elements of these 4 to be zero. Figure 1 shows how this could look like.

Figure 1: Structure supported by the sparse matrix multiplication feature in Ampere GPUs. The figure is taken from Jeff Pool’s GTC 2020 presentation on [Accelerating Sparsity in the NVIDIA Ampere Architecture](https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22085-accelerating-sparsity-in-the-nvidia-ampere-architecture%E2%80%8B.pdf) by the courtesy of NVIDIA.

When you multiply this sparse weight matrix with some dense inputs, the sparse matrix tensor core feature in Ampere automatically compresses the sparse matrix to a dense representation that is half the size as can be seen in Figure 2. After this compression, the densely compressed matrix tile is fed into the tensor core which computes a matrix multiplication of twice the usual size. This effectively yields a 2x speedup since the bandwidth requirements during matrix multiplication from shared memory are halved.

Figure 2: The sparse matrix is compressed to a dense representation before the matrix multiplication is performed. The figure is taken from Jeff Pool’s GTC 2020 presentation on [Accelerating Sparsity in the NVIDIA Ampere Architecture](https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s22085-accelerating-sparsity-in-the-nvidia-ampere-architecture%E2%80%8B.pdf) by the courtesy of NVIDIA.

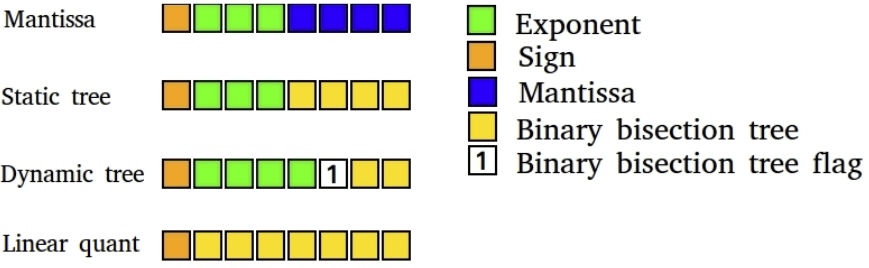
I was working on [sparse network training](https://arxiv.org/abs/1907.04840) in my research and I also wrote a [blog post about sparse training](https://timdettmers.com/2019/07/11/sparse-networks-from-scratch/). One criticism of my work was that “You reduce the FLOPS required for the network, but it does not yield speedups because GPUs cannot do fast sparse matrix multiplication.” Well, with the addition of the sparse matrix multiplication feature for Tensor Cores, my algorithm, or [other](https://arxiv.org/abs/2002.03231) [sparse](https://arxiv.org/abs/2002.07376) [training](https://arxiv.org/abs/1911.11134) [algorithms](https://arxiv.org/abs/1902.05967), now actually provide speedups of up to 2x during training.

[](https://arxiv.org/abs/1907.04840)Figure 3: The [sparse training algorithm](https://arxiv.org/abs/1907.04840) that I developed has three stages: (1) Determine the importance of each layer. (2) Remove the smallest, unimportant weights. (3) Grow new weights proportional to the importance of each layer. Read more about my work in my [sparse training blog post](https://timdettmers.com/2019/07/11/sparse-networks-from-scratch/).

While this feature is still experimental and training sparse networks are not commonplace yet, having this feature on your GPU means you are ready for the future of sparse training.

Low-precision Computation

In my work, I’ve previously shown that new data types can improve stability during [low-precision backpropagation](https://arxiv.org/abs/1511.04561).

Figure 4: Low-precision deep learning 8-bit datatypes that I developed. Deep learning training benefits from highly specialized data types. My dynamic tree datatype uses a dynamic bit that indicates the beginning of a binary bisection tree that quantized the range [0, 0.9] while all previous bits are used for the exponent. This allows to dynamically represent numbers that are both large and small with high precision.

Currently, if you want to have stable backpropagation with 16-bit floating-point numbers (FP16), the big problem is that ordinary FP16 data types only support numbers in the range [-65,504, 65,504]. If your gradient slips past this range, your gradients explode into NaN values. To prevent this during FP16 training, we usually perform loss scaling where you multiply the loss by a small number before backpropagating to prevent this gradient explosion.

The BrainFloat 16 format (BF16) uses more bits for the exponent such that the range of possible numbers is the same as for FP32: [-3\*10^38, 3\*10^38]. BF16 has less precision, that is significant digits, but gradient precision is not that important for learning. So what BF16 does is that you no longer need to do any loss scaling or worry about the gradient blowing up quickly. As such, we should see an increase in training stability by using the BF16 format as a slight loss of precision.

What this means for you: With BF16 precision, training might be more stable than with FP16 precision while providing the same speedups. With 32-bit TensorFloat (TF32) precision, you get near FP32 stability while giving the speedups close to FP16. The good thing is, to use these data types, you can just replace FP32 with TF32 and FP16 with BF16 — no code changes required!

Overall, though, these new data types can be seen as lazy data types in the sense that you could have gotten all the benefits with the old data types with some additional programming efforts (proper loss scaling, initialization, normalization, using Apex). As such, these data types do not provide speedups but rather improve ease of use of low precision for training.

Fan Designs and GPUs Temperature Issues

While the new fan design of the RTX 30 series performs very well to cool the GPU, different fan designs of non-founders edition GPUs might be more problematic. If your GPU heats up beyond 80C, it will throttle itself and slow down its computational speed / power. This overheating can happen in particular if you stack multiple GPUs next to each other. A solution to this is to use PCIe extenders to create space between GPUs.

Spreading GPUs with PCIe extenders is very effective for cooling, and other fellow PhD students at the University of Washington and I use this setup with great success. It does not look pretty, but it keeps your GPUs cool! This has been running with no problems at all for 4 years now. It can also help if you do not have enough space to fit all GPUs in the PCIe slots. For example, if you can find the space within a desktop computer case, it might be possible to buy standard 3-slot-width RTX 4090 and spread them with PCIe extenders within the case. With this, you might solve both the space issue and cooling issue for a 4x RTX 4090 setup with a single simple solution.



Figure 5: 4x GPUs with PCIe extenders. It looks like a mess, but it is very effective for cooling. I used this rig for 4 years and cooling is excellent despite problematic RTX 2080 Ti Founders Edition GPUs.

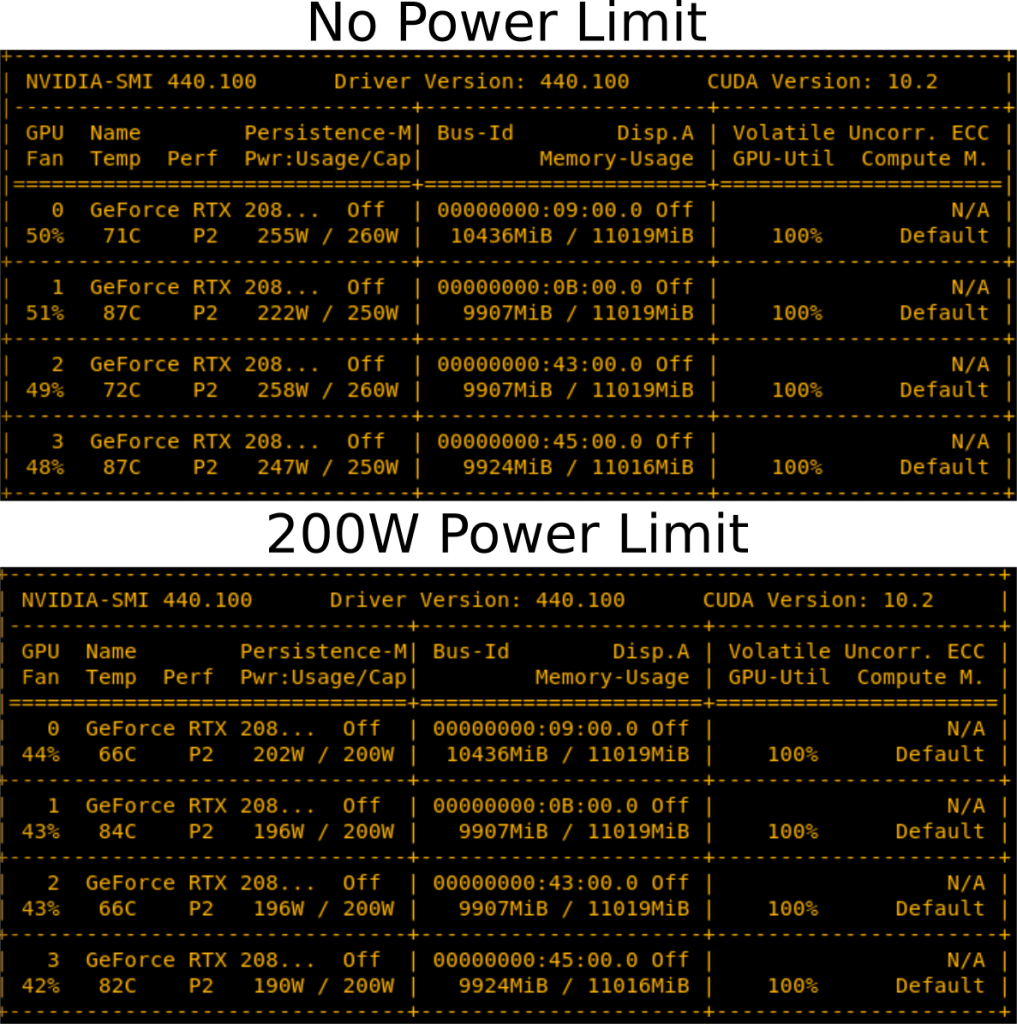
3-slot Design and Power Issues

The RTX 3090 and RTX 4090 are 3-slot GPUs, so one will not be able to use it in a 4x setup with the default fan design from NVIDIA. This is kind of justified because it runs at over 350W TDP, and it will be difficult to cool in a multi-GPU 2-slot setting. The RTX 3080 is only slightly better at 320W TDP, and cooling a 4x RTX 3080 setup will also be very difficult.

It is also difficult to power a 4x 350W = 1400W or 4x 450W = 1800W system in the 4x RTX 3090 or 4x RTX 4090 case. Power supply units (PSUs) of 1600W are readily available, but having only 200W to power the [CPU and motherboard](https://timdettmers.com/2018/12/16/deep-learning-hardware-guide/) can be too tight. The components’ maximum power is only used if the components are fully utilized, and in deep learning, the CPU is usually only under weak load. With that, a 1600W PSU might work quite well with a 4x RTX 3080 build, but for a 4x RTX 3090 build, it is better to look for high wattage PSUs (+1700W). Some of my followers have had great success with cryptomining PSUs — have a look in the comment section for more info about that. Otherwise, it is important to note that not all outlets support PSUs above 1600W, especially in the US. This is the reason why in the US, there are currently few standard desktop PSUs above 1600W on the market. If you get a server or cryptomining PSUs, beware of the form factor — make sure it fits into your computer case.

Power Limiting: An Elegant Solution to Solve the Power Problem?

It is possible to set a power limit on your GPUs. So you would be able to programmatically set the power limit of an RTX 3090 to 300W instead of their standard 350W. In a 4x GPU system, that is a saving of 200W, which might just be enough to build a 4x RTX 3090 system with a 1600W PSU feasible. It also helps to keep the GPUs cool. So setting a power limit can solve the two major problems of a 4x RTX 3080 or 4x RTX 3090 setups, cooling, and power, at the same time. For a 4x setup, you still need effective blower GPUs (and the standard design may prove adequate for this), but this resolves the PSU problem.

Figure 6: Reducing the power limit has a slight cooling effect. Reducing the RTX 2080 Ti power limit by 50-60 W decreases temperatures slightly and fans run more silent.

You might ask, “Doesn’t this slow down the GPU?” Yes, it does, but the question is by how much. I benchmarked the 4x RTX 2080 Ti system shown in Figure 5 under different power limits to test this. I benchmarked the time for 500 mini-batches for BERT Large during inference (excluding the softmax layer). I choose BERT Large inference since, from my experience, this is the deep learning model that stresses the GPU the most. As such, I would expect power limiting to have the most massive slowdown for this model. As such, the slowdowns reported here are probably close to the maximum slowdowns that you can expect. The results are shown in Figure 7.

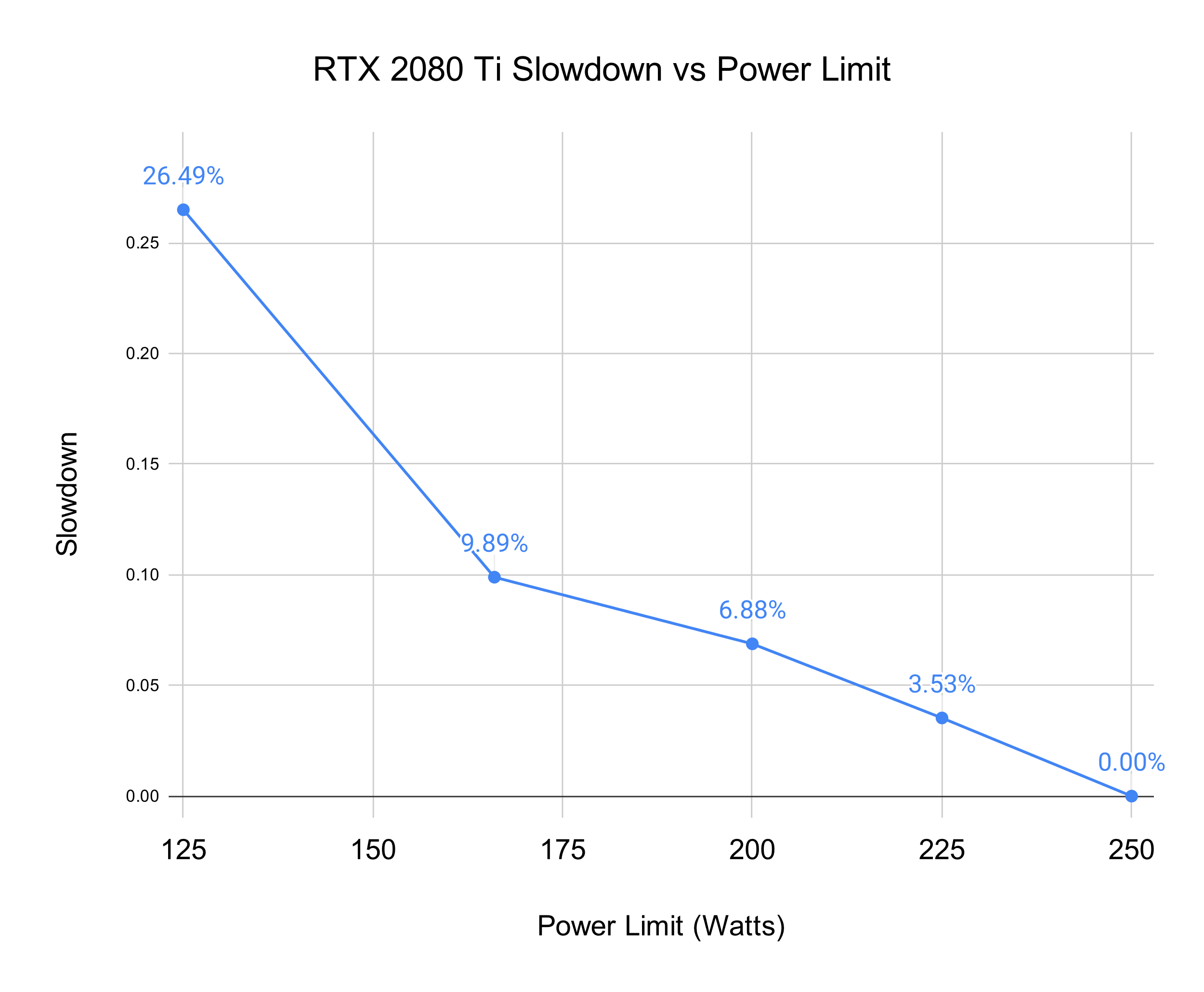


Figure 7: Measured slowdown for a given power limit on an RTX 2080 Ti. Measurements taken are mean processing times for 500 mini-batches of BERT Large during inference (excluding softmax layer).

As we can see, setting the power limit does not seriously affect performance. Limiting the power by 50W — more than enough to handle 4x RTX 3090 — decreases performance by only 7%.

RTX 4090s and Melting Power Connectors: How to Prevent Problems

There was a misconception that RTX 4090 power cables melt because they were bent. However, it was found that only 0.1% of users had this problem and the problem occured due to user error. Here a video that shows that the main problem is that [cables were not inserted correctly](https://www.youtube.com/watch?v=ig2px7ofKhQ&t=1065s).

So using RTX 4090 cards is perfectly safe if you follow the following install instructions:

1. If you use an old cable or old GPU make sure the contacts are free of debri / dust.
2. Use the power connector and stick it into the socket until you hear a \*click\* — this is the most important part.
3. Test for good fit by wiggling the power cable left to right. The cable should not move.
4. Check the contact with the socket visually, there should be no gap between cable and socket.

8-bit Float Support in H100 and RTX 40 series GPUs

The support of the 8-bit Float (FP8) is a huge advantage for the RTX 40 series and H100 GPUs. With 8-bit inputs it allows you to load the data for matrix multiplication twice as fast, you can store twice as much matrix elements in your caches which in the Ada and Hopper architecture are very large, and now with FP8 tensor cores you get 0.66 PFLOPS of compute for a RTX 4090 — this is more FLOPS then the entirety of the worlds fastest supercomputer in year 2007. 4x RTX 4090 with FP8 compute rival the faster supercomputer in the world in year 2010 (deep learning started to work just in 2009).

The main problem with using 8-bit precision is that transformers can get very unstable with so few bits and crash during training or generate non-sense during inference. I have written a [paper about the emergence of instabilities in large language models](https://arxiv.org/abs/2208.07339) and I also written a more accessible [blog post](https://timdettmers.com/2022/08/17/llm-int8-and-emergent-features/).

The main take-way is this: Using 8-bit instead of 16-bit makes things very unstable, but if you keep a couple of dimensions in high precision everything works just fine.

[A graph with numbers and lines

Description automatically generated](https://i0.wp.com/timdettmers.com/wp-content/uploads/2023/01/LLM_int8_zeroshot_emergence.png?ssl=1)Main results from my work on 8-bit matrix multiplication for Large Language Models (LLMs). We can see that the best 8-bit baseline fails to deliver good zero-shot performance. The method that I developed, LLM.int8(), can perform Int8 matrix multiplication with the same results as the 16-bit baseline.

But Int8 was already supported by the RTX 30 / A100 / Ampere generation GPUs, why is FP8 in the RTX 40 another big upgrade? The FP8 data type is much more stable than the Int8 data type and its easy to use it in functions like layer norm or non-linear functions, which are difficult to do with Integer data types. This will make it very straightforward to use it in training and inference. I think this will make FP8 training and inference relatively common in a couple of months.

If you want to read more about the advantages of Float vs Integer data types you can read my recent paper about [k-bit inference scaling laws](https://arxiv.org/abs/2212.09720). Below you can see one relevant main result for Float vs Integer data types from this paper. We can see that bit-by-bit, the FP4 data type preserve more information than Int4 data type and thus improves the mean LLM zeroshot accuracy across 4 tasks.

[A graph with different colored lines

Description automatically generated](https://i0.wp.com/timdettmers.com/wp-content/uploads/2023/01/pythia_4bit_datatypes2.png?ssl=1)4-bit Inference scaling laws for Pythia Large Language Models for different data types. We see that bit-by-bit, 4-bit float data types have better zeroshot accuracy compared to the Int4 data types.

Raw Performance Ranking of GPUs

Below we see a chart of raw relevative performance across all GPUs. We see that there is a gigantic gap in 8-bit performance of H100 GPUs and old cards that are optimized for 16-bit performance.

[A graph with numbers and lines

Description automatically generated with medium confidence](https://i0.wp.com/timdettmers.com/wp-content/uploads/2023/01/GPUS_Ada_raw_performance3.png?ssl=1)Shown is raw relative transformer performance of GPUs. For example, an RTX 4090 has about 0.33x performance of a H100 SMX for 8-bit inference. In other words, a H100 SMX is three times faster for 8-bit inference compared to a RTX 4090.

For this data, I did not model 8-bit compute for older GPUs. I did so, because 8-bit Inference and training are much more effective on Ada/Hopper GPUs because of the 8-bit Float data type and Tensor Memory Accelerator (TMA) which saves the overhead of computing read/write indices which is particularly helpful for 8-bit matrix multiplication. Ada/Hopper also have FP8 support, which makes in particular 8-bit training much more effective.

I did not model numbers for 8-bit training because to model that I need to know the latency of L1 and L2 caches on Hopper/Ada GPUs, and they are unknown and I do not have access to such GPUs. On Hopper/Ada, 8-bit training performance can well be 3-4x of 16-bit training performance if the caches are as fast as rumored.

But even with the new FP8 tensor cores there are some additional issues which are difficult to take into account when modeling GPU performance. For example, FP8 tensor cores do not support transposed matrix multiplication which means backpropagation needs either a separate transpose before multiplication or one needs to hold two sets of weights — one transposed and one non-transposed — in memory. I used two sets of weight when I experimented with Int8 training in my [LLM.int8()](https://arxiv.org/abs/2208.07339) project and this reduced the overall speedups quite significantly. I think one can do better with the right algorithms/software, but this shows that missing features like a transposed matrix multiplication for tensor cores can affect performance.

For old GPUs, Int8 inference performance is close to the 16-bit inference performance for models below 13B parameters. Int8 performance on old GPUs is only relevant if you have relatively large models with 175B parameters or more. If you are interested in 8-bit performance of older GPUs, you can read the Appendix D of my [LLM.int8() paper](https://arxiv.org/abs/2208.07339) where I benchmark Int8 performance.

GPU Deep Learning Performance per Dollar

Below we see the chart for the performance per US dollar for all GPUs sorted by 8-bit inference performance. How to use the chart to find a suitable GPU for you is as follows:

1. Determine the amount of GPU memory that you need (rough heuristic: at least 12 GB for image generation; at least 24 GB for work with transformers)
2. While 8-bit inference and training is experimental, it will become standard within 6 months. You might need to do some extra difficult coding to work with 8-bit in the meantime. Is that OK for you? If not, select for 16-bit performance.
3. Using the metric determined in (2), find the GPU with the highest relative performance/dollar that has the amount of memory you need.

We can see that the RTX 4070 Ti is most cost-effective for 8-bit and 16-bit inference while the RTX 3080 remains most cost-effective for 16-bit training. While these GPUs are most cost-effective, they are not necessarily recommended as they do not have sufficient memory for many use-cases. However, it might be the ideal cards to get started on your deep learning journey. Some of these GPUs are excellent for Kaggle competition where one can often rely on smaller models. Since to do well in Kaggle competitions the method of how you work is more important than the models size, many of these smaller GPUs are excellent for Kaggle competitions.

The best GPUs for academic and startup servers seem to be A6000 Ada GPUs (not to be confused with A6000 Turing). The H100 SXM GPU is also very cost effective and has high memory and very strong performance. If I would build a small cluster for a company/academic lab, I would use 66-80% A6000 GPUs and 20-33% H100 SXM GPUs. If I get a good deal on L40 GPUs, I would also pick them instead of A6000, so you can always ask for a quote on these.

[A graph of a number of numbers

Description automatically generated with medium confidence](https://i0.wp.com/timdettmers.com/wp-content/uploads/2023/01/GPUs_Ada_performance_per_dollar6.png?ssl=1)Shown is relative performance per US Dollar of GPUs normalized by the cost for a desktop computer and the average Amazon and eBay price for each GPU. Additionally, the electricity cost of ownership for 5 years is added with an electricity price of 0.175 USD per kWh and a 15% GPU utilization rate. The electricity cost for a RTX 4090 is about $100 per year. How to read and interpret the chart: a desktop computer with RTX 4070 Ti cards owned for 5 years yields about 2x more 8-bit inference performance per dollar compared to a RTX 3090 GPU.

GPU Recommendations

I have a create a recommendation flow-chart that you can see below (click here for [interactive app](https://nanx.me/gpu/) from Nan Xiao). While this chart will help you in 80% of cases, it might not quite work for you because the options might be too expensive. In that case, try to look at the benchmarks above and pick the most cost effective GPU that still has enough GPU memory for your use-case. You can estimate the GPU memory needed by running your problem in the vast.ai or Lambda Cloud for a while so you know what you need. The vast.ai or Lambda Cloud might also work well if you only need a GPU very sporadically (every couple of days for a few hours) and you do not need to download and process large dataset to get started. However, cloud GPUs are usually not a good option if you use your GPU for many months with a high usage rate each day (12 hours each day). You can use the example in the “When is it better to use the cloud vs a dedicated GPU desktop/server?” section below to determine if cloud GPUs are good for you.

[A diagram of a diagram

Description automatically generated](https://i0.wp.com/timdettmers.com/wp-content/uploads/2023/01/gpu_recommendations.png?ssl=1)GPU recommendation chart for Ada/Hopper GPUs. Follow the answers to the Yes/No questions to find the GPU that is most suitable for you. While this chart works well in about 80% of cases, you might end up with a GPU that is too expensive. Use the cost/performance charts above to make a selection instead. [[interactive app](https://nanx.me/gpu/)]

Is it better to wait for future GPUs for an upgrade? The future of GPUs.

To understand if it makes sense to skip this generation and buy the next generation of GPUs, it makes sense to talk a bit about what improvements in the future will look like.

In the past it was possible to shrink the size of transistors to improve speed of a processor. This is coming to an end now. For example, while shrinking SRAM increased its speed (smaller distance, faster memory access), this is no longer the case. Current improvements in SRAM do not improve its performance anymore and might even be negative. While logic such as Tensor Cores get smaller, this does not necessarily make GPU faster since the main problem for matrix multiplication is to get memory to the tensor cores which is dictated by SRAM and GPU RAM speed and size. GPU RAM still increases in speed if we stack memory modules into high-bandwidth modules (HBM3+), but these are too expensive to manufacture for consumer applications. The main way to improve raw speed of GPUs is to use more power and more cooling as we have seen in the RTX 30s and 40s series. But this cannot go on for much longer.

Chiplets such as used by AMD CPUs are another straightforward way forward. AMD beat Intel by developing CPU chiplets. Chiplets are small chips that are fused together with a high speed on-chip network. You can think about them as two GPUs that are so physically close together that you can almost consider them a single big GPU. They are cheaper to manufacture, but more difficult to combine into one big chip. So you need know-how and fast connectivity between chiplets. AMD has a lot of experience with chiplet design. AMD’s next generation GPUs are going to be chiplet designs, while NVIDIA currently has no public plans for such designs. This may mean that the next generation of AMD GPUs might be better in terms of cost/performance compared to NVIDIA GPUs.

However, the main performance boost for GPUs is currently specialized logic. For example, the asynchronous copy hardware units on the Ampere generation (RTX 30 / A100 / RTX 40) or the extension, the Tensor Memory Accelerator (TMA), both reduce the overhead of copying memory from the slow global memory to fast shared memory (caches) through specialized hardware and so each thread can do more computation. The TMA also reduces overhead by performing automatic calculations of read/write indices which is particularly important for 8-bit computation where one has double the elements for the same amount of memory compared to 16-bit computation. So specialized hardware logic can accelerate matrix multiplication further.  
Low-bit precision is another straightforward way forward for a couple of years. We will see widespread adoption of 8-bit inference and training in the next months. We will see widespread 4-bit inference in the next year. Currently, the technology for 4-bit training does not exists, but research looks promising and I expect the first high performance FP4 Large Language Model (LLM) with competitive predictive performance to be trained in 1-2 years time.

Going to 2-bit precision for training currently looks pretty impossible, but it is a much easier problem than shrinking transistors further. So progress in hardware mostly depends on software and algorithms that make it possible to use specialized features offered by the hardware.

We will probably be able to still improve the combination of algorithms + hardware to the year 2032, but after that will hit the end of GPU improvements (similar to smartphones). The wave of performance improvements after 2032 will come from better networking algorithms and mass hardware. It is uncertain if consumer GPUs will be relevant at this point. It might be that you need an RTX 9090 to run run Super HyperStableDiffusion Ultra Plus 9000 Extra or OpenChatGPT 5.0, but it might also be that some company will offer a high-quality API that is cheaper than the electricity cost for a RTX 9090 and you want to use a laptop + API for image generation and other tasks.

Overall, I think investing into a 8-bit capable GPU will be a very solid investment for the next 9 years. Improvements at 4-bit and 2-bit are likely small and other features like Sort Cores would only become relevant once sparse matrix multiplication can be leveraged well. We will probably see some kind of other advancement in 2-3 years which will make it into the next GPU 4 years from now, but we are running out of steam if we keep relying on matrix multiplication. This makes investments into new GPUs last longer.

Question & Answers & Misconceptions

Do I need PCIe 4.0 or PCIe 5.0?

Generally, no. PCIe 5.0 or 4.0 is great if you have a GPU cluster. It is okay if you have an 8x GPU machine, but otherwise, it does not yield many benefits. It allows better parallelization and a bit faster data transfer. Data transfers are not a bottleneck in any application. In computer vision, in the data transfer pipeline, the data storage can be a bottleneck, but not the PCIe transfer from CPU to GPU. So there is no real reason to get a PCIe 5.0 or 4.0 setup for most people. The benefits will be maybe 1-7% better parallelization in a 4 GPU setup.

Do I need 8x/16x PCIe lanes?

Same as with PCIe 4.0 — generally, no. [PCIe lanes](https://timdettmers.com/2018/12/16/deep-learning-hardware-guide/) are needed for parallelization and fast data transfers, which are seldom a bottleneck. Operating GPUs on 4x lanes is fine, especially if you only have 2 GPUs. For a 4 GPU setup, I would prefer 8x lanes per GPU, but running them at 4x lanes will probably only decrease performance by around 5-10% if you parallelize across all 4 GPUs.

How do I fit 4x RTX 4090 or 3090 if they take up 3 PCIe slots each?

You need to get one of the two-slot variants, or you can try to spread them out with PCIe extenders. Besides space, you should also immediately think about cooling and a suitable PSU.

PCIe extenders might also solve both space and cooling issues, but you need to make sure that you have enough space in your case to spread out the GPUs. Make sure your PCIe extenders are long enough!

How do I cool 4x RTX 3090 or 4x RTX 3080?

See the previous section.

Can I use multiple GPUs of different GPU types?

Yes, you can! But you cannot parallelize efficiently across GPUs of different types since you will often go at the speed of the slowest GPU (data and fully sharded parallelism). So different GPUs work just fine, but parallelization across those GPUs will be inefficient since the fastest GPU will wait for the slowest GPU to catch up to a synchronization point (usually gradient update).

What is NVLink, and is it useful?

Generally, NVLink is not useful. NVLink is a high speed interconnect between GPUs. It is useful if you have a GPU cluster with +128 GPUs. Otherwise, it yields almost no benefits over standard PCIe transfers.

I do not have enough money, even for the cheapest GPUs you recommend. What can I do?

Definitely buy used GPUs. You can buy a small cheap GPU for prototyping and testing and then roll out for full experiments to the cloud like vast.ai or Lambda Cloud. This can be cheap if you train/fine-tune/inference on large models only every now and then and spent more time protoyping on smaller models.

What is the carbon footprint of GPUs? How can I use GPUs without polluting the environment?

I built a [carbon calculator](https://github.com/TimDettmers/carbonneutral) for calculating your carbon footprint for academics (carbon from flights to conferences + GPU time). The calculator can also be used to calculate a pure GPU carbon footprint. You will find that GPUs produce much, much more carbon than international flights. As such, you should make sure you have a green source of energy if you do not want to have an astronomical carbon footprint. If no electricity provider in our area provides green energy, the best way is to buy carbon offsets. Many people are skeptical about carbon offsets. Do they work? Are they scams?

I believe skepticism just hurts in this case, because not doing anything would be more harmful than risking the probability of getting scammed. If you worry about scams, just invest in a portfolio of offsets to minimize risk.

I worked on a project that produced carbon offsets about ten years ago. The carbon offsets were generated by burning leaking methane from mines in China. UN officials tracked the process, and they required clean digital data and physical inspections of the project site. In that case, the carbon offsets that were produced were highly reliable. I believe many other projects have similar quality standards.

What do I need to parallelize across two machines?

If you want to be on the safe side, you should get at least +50Gbits/s network cards to gain speedups if you want to parallelize across machines. I recommend having at least an EDR Infiniband setup, meaning a network card with at least 50 GBit/s bandwidth. Two EDR cards with cable are about $500 on eBay.

In some cases, you might be able to get away with 10 Gbit/s Ethernet, but this is usually only the case for special networks (certain convolutional networks) or if you use certain algorithms (Microsoft DeepSpeed).

Is the sparse matrix multiplication features suitable for sparse matrices in general?

It does not seem so. Since the granularity of the sparse matrix needs to have 2 zero-valued elements, every 4 elements, the sparse matrices need to be quite structured. It might be possible to adjust the algorithm slightly, which involves that you pool 4 values into a compressed representation of 2 values, but this also means that precise arbitrary sparse matrix multiplication is not possible with Ampere GPUs.

Do I need an Intel CPU to power a multi-GPU setup?

I do not recommend Intel CPUs unless you heavily use CPUs in Kaggle competitions (heavy linear algebra on the CPU). Even for Kaggle competitions AMD CPUs are still great, though. AMD CPUs are cheaper and better than Intel CPUs in general for deep learning. For a 4x GPU built, my go-to CPU would be a Threadripper. We built dozens of systems at our university with Threadrippers, and they all work great — no complaints yet. For 8x GPU systems, I would usually go with CPUs that your vendor has experience with. CPU and PCIe/system reliability is more important in 8x systems than straight performance or straight cost-effectiveness.

Does computer case design matter for cooling?

No. GPUs are usually perfectly cooled if there is at least a small gap between GPUs. Case design will give you 1-3 C better temperatures, space between GPUs will provide you with 10-30 C improvements. The bottom line, if you have space between GPUs, cooling does not matter. If you have no space between GPUs, you need the right cooler design (blower fan) or another solution (water cooling, PCIe extenders), but in either case, case design and case fans do not matter.

Will AMD GPUs + ROCm ever catch up with NVIDIA GPUs + CUDA?

Not in the next 1-2 years. It is a three-way problem: Tensor Cores, software, and community.

AMD GPUs are great in terms of pure silicon: Great FP16 performance, great memory bandwidth. However, their lack of Tensor Cores or the equivalent makes their deep learning performance poor compared to NVIDIA GPUs. Packed low-precision math does not cut it. Without this hardware feature, AMD GPUs will never be competitive. Rumors show that [some data center card](https://wccftech.com/amd-cdna-architecture-radeon-instinct-arcturus-gpu-120-cu-7680-cores/) with Tensor Core equivalent is planned for 2020, but no new data emerged since then. Just having data center cards with a Tensor Core equivalent would also mean that few would be able to afford such AMD GPUs, which would give NVIDIA a competitive advantage.

Let’s say AMD introduces a Tensor-Core-like-hardware feature in the future. Then many people would say, “But there is no software that works for AMD GPUs! How am I supposed to use them?” This is mostly a misconception. The AMD software via ROCm has come to a long way, and support via PyTorch is excellent. While I have not seen many experience reports for AMD GPUs + PyTorch, all the software features are integrated. It seems, if you pick any network, you will be just fine running it on AMD GPUs. So here AMD has come a long way, and this issue is more or less solved.

However, if you solve software and the lack of Tensor Cores, AMD still has a problem: the lack of community. If you have a problem with NVIDIA GPUs, you can Google the problem and find a solution. That builds a lot of trust in NVIDIA GPUs. You have the infrastructure that makes using NVIDIA GPUs easy (any deep learning framework works, any scientific problem is well supported). You have the hacks and tricks that make usage of NVIDIA GPUs a breeze (e.g., apex). You can find experts on NVIDIA GPUs and programming around every other corner while I knew much less AMD GPU experts.

In the community aspect, AMD is a bit like Julia vs Python. Julia has a lot of potential, and many would say, and rightly so, that it is the superior programming language for scientific computing. Yet, Julia is barely used compared to Python. This is because the Python community is very strong. Numpy, SciPy, Pandas are powerful software packages that a large number of people congregate around. This is very similar to the NVIDIA vs AMD issue.

Thus, it is likely that AMD will not catch up until Tensor Core equivalent is introduced (1/2 to 1 year?) and a strong community is built around ROCm (2 years?). AMD will always snatch a part of the market share in specific subgroups (e.g., cryptocurrency mining, data centers). Still, in deep learning, NVIDIA will likely keep its monopoly for at least a couple more years.

When is it better to use the cloud vs a dedicated GPU desktop/server?

Rule-of-thumb: If you expect to do deep learning for longer than a year, it is cheaper to get a desktop GPU. Otherwise, cloud instances are preferable unless you have extensive cloud computing skills and want the benefits of scaling the number of GPUs up and down at will.

Numbers in the following paragraphs are going to change, but it serves as a scenario that helps you to understand the rough costs. You can use similar math to determine if cloud GPUs are the best solution for you.

For the exact point in time when a cloud GPU is more expensive than a desktop depends highly on the service that you are using, and it is best to do a little math on this yourself. Below I do an example calculation for an AWS V100 spot instance with 1x V100 and compare it to the price of a desktop with a single RTX 3090 (similar performance). The desktop with RTX 3090 costs $2,200 (2-GPU barebone + RTX 3090). Additionally, assuming you are in the US, there is an additional $0.12 per kWh for electricity. This compares to $2.14 per hour for the AWS on-demand instance.

At 15% utilization per year, the desktop uses:

(350 W (GPU) + 100 W (CPU))\*0.15 (utilization) \* 24 hours \* 365 days = 591 kWh per year

So 591 kWh of electricity per year, that is an additional $71.

The break-even point for a desktop vs a cloud instance at 15% utilization (you use the cloud instance 15% of time during the day), would be about 300 days ($2,311 vs $2,270):

$2.14/h \* 0.15 (utilization) \* 24 hours \* 300 days = $2,311

So if you expect to run deep learning models after 300 days, it is better to buy a desktop instead of using AWS on-demand instances.

You can do similar calculations for any cloud service to make the decision if you go for a cloud service or a desktop.

Common utilization rates are the following:

* PhD student personal desktop: < 15%
* PhD student slurm GPU cluster: > 35%
* Company-wide slurm research cluster: > 60%

In general, utilization rates are lower for professions where thinking about cutting edge ideas is more important than developing practical products. Some areas have low utilization rates (interpretability research), while other areas have much higher rates (machine translation, language modeling). In general, the utilization of personal machines is almost always overestimated. Commonly, most personal systems have a utilization rate between 5-10%. This is why I would highly recommend slurm GPU clusters for research groups and companies instead of individual desktop GPU machines.

Version History

* 2023-01-30: Improved font and recommendation chart. Added 5 years cost of ownership electricity perf/USD chart. Updated Async copy and TMA functionality. Slight update to FP8 training. General improvements.
* 2023-01-16: Added Hopper and Ada GPUs. Added GPU recommendation chart. Added information about the TMA unit and L2 cache.
* 2020-09-20: Added discussion of using power limiting to run 4x RTX 3090 systems. Added older GPUs to the performance and cost/performance charts. Added figures for sparse matrix multiplication.
* 2020-09-07: Added NVIDIA Ampere series GPUs. Included lots of good-to-know GPU details.
* 2019-04-03: Added RTX Titan and GTX 1660 Ti. Updated TPU section. Added startup hardware discussion.
* 2018-11-26: Added discussion of overheating issues of RTX cards.
* 2018-11-05: Added RTX 2070 and updated recommendations. Updated charts with hard performance data. Updated TPU section.
* 2018-08-21: Added RTX 2080 and RTX 2080 Ti; reworked performance analysis
* 2017-04-09: Added cost-efficiency analysis; updated recommendation with NVIDIA Titan Xp
* 2017-03-19: Cleaned up blog post; added GTX 1080 Ti
* 2016-07-23: Added Titan X Pascal and GTX 1060; updated recommendations
* 2016-06-25: Reworked multi-GPU section; removed simple neural network memory section as no longer relevant; expanded convolutional memory section; truncated AWS section due to not being efficient anymore; added my opinion about the Xeon Phi; added updates for the GTX 1000 series
* 2015-08-20: Added section for AWS GPU instances; added GTX 980 Ti to the comparison relation
* 2015-04-22: GTX 580 no longer recommended; added performance relationships between cards
* 2015-03-16: Updated GPU recommendations: GTX 970 and GTX 580
* 2015-02-23: Updated GPU recommendations and memory calculations
* 2014-09-28: Added emphasis for memory requirement of CNNs

Acknowledgments

I thank Suhail for making me aware of outdated prices on H100 GPUs, Gjorgji Kjosev for pointing out font issues, Anonymous for pointing out that the TMA unit does not exist on Ada GPUs, Scott Gray for pointing out that FP8 tensor cores have no transposed matrix multiplication, and reddit and HackerNews users for pointing out many other improvements.

For past updates of this blog post, I want to thank Mat Kelcey for helping me to debug and test custom code for the GTX 970; I want to thank Sander Dieleman for making me aware of the shortcomings of my GPU memory advice for convolutional nets; I want to thank Hannes Bretschneider for pointing out software dependency problems for the GTX 580; and I want to thank Oliver Griesel for pointing out notebook solutions for AWS instances. I want to thank Brad Nemire for providing me with an RTX Titan for benchmarking purposes. I want to thank Agrin Hilmkil, Ari Holtzman, Gabriel Ilharco, Nam Pho for their excellent feedback on the previous version of this blog post.